**Group 10: Srinivas Bhootam, Shashi Vadhan, Imran Shaik**

Srinivas Bhootam

DeVos Graduate School, Northwood University

MGT-659-NW2 Relational Database Management

David Sanford

05/10/2025

**Smart City Utilities Database**

**Description:**  
Smart City Utilities Management system and their relationships using primary and foreign keys.

**Entities & Relationships:**

* **Residents** → resident\_id (PK)
* **Utility\_Types** → utility\_id (PK)
* **Bills** → bill\_id (PK), with FKs referencing resident\_id and utility\_id
* **Utility\_Usage** → usage\_id (PK), with FK referencing bill\_id

**Relationships:**

* One Resident → Many Bills
* One Utility Type → Many Bills
* One Bill → One Utility Usage

**Step 2: SQL Scripts (Database & Tables)**

**Description:**  
Below is the SQL code used to create the smart\_city\_utilities database and its four tables with appropriate keys and constraints.  
  
CREATE DATABASE smart\_city\_utilities;

CREATE TABLE Residents (

resident\_id SERIAL PRIMARY KEY,

name VARCHAR (100),

address TEXT,

phone\_number VARCHAR (15)

);

CREATE TABLE Utility\_Types (

utility\_id SERIAL PRIMARY KEY,

utility\_name VARCHAR (50)

);

CREATE TABLE Bills (

bill\_id SERIAL PRIMARY KEY,

resident\_id INT,

utility\_id INT,

billing\_date DATE,

amount DECIMAL (10, 2),

FOREIGN KEY (resident\_id) REFERENCES Residents(resident\_id),

FOREIGN KEY (utility\_id) REFERENCES Utility\_Types(utility\_id)

);

CREATE TABLE Utility\_Usage (

usage\_id SERIAL PRIMARY KEY,

bill\_id INT,

usage\_amount DECIMAL (10, 2),

unit VARCHAR (20),

FOREIGN KEY (bill\_id) REFERENCES Bills(bill\_id)

);

**Step 3: Types of queries and views explored on created database**

**1. Resident Utility Consumption Summary (Monthly):**

SELECT

r. resident\_id,

r.name,

DATE\_TRUNC ('month', b. billing\_date) AS month,

ut.utility\_name,

SUM(u.usage\_amount) AS total\_usage,

SUM(b.amount) AS total\_amount

FROM Residents r

JOIN Bills b ON r.resident\_id = b.resident\_id

JOIN Utility\_Types ut ON b.utility\_id = ut.utility\_id

JOIN Utility\_Usage u ON b.bill\_id = u.bill\_id

GROUP BY r.resident\_id, r.name, month, ut.utility\_name

ORDER BY month, r.name;

**A screenshot of a computer

AI-generated content may be incorrect.**

**2. Top 5 Spenders:**

SELECT

r.name,

SUM(b.amount) AS total\_spent

FROM Residents r

JOIN Bills b ON r.resident\_id = b.resident\_id

GROUP BY r.name

ORDER BY total\_spent DESC

LIMIT 5;

A screenshot of a computer

AI-generated content may be incorrect.

**3. Utility Type Usage Trend:**

SELECT

ut.utility\_name,

DATE\_TRUNC ('month', b.billing\_date) AS month,

SUM(u.usage\_amount) AS total\_usage

FROM Utility\_Types ut

JOIN Bills b ON ut.utility\_id = b.utility\_id

JOIN Utility\_Usage u ON b.bill\_id = u.bill\_id

GROUP BY ut.utility\_name, month

ORDER BY ut.utility\_name, month;

A screenshot of a data output

AI-generated content may be incorrect.

**4. Total Bill for Each Resident:**

SELECT

r.name,

COUNT(b.bill\_id) AS total\_bills,

SUM(b.amount) AS total\_amount

FROM Residents r

JOIN Bills b ON r.resident\_id = b.resident\_id

GROUP BY r.name;

A screenshot of a computer

AI-generated content may be incorrect.

**Step 4: Useful Views explored**

**(i) vw\_monthly\_usage\_summary**

CREATE VIEW vw\_monthly\_usage\_summary AS

SELECT

r.resident\_id, r.name,

DATE\_TRUNC('month', b.billing\_date) AS month,

ut.utility\_name,

SUM(u.usage\_amount) AS total\_usage,

SUM(b.amount) AS total\_amount

FROM Residents r

JOIN Bills b ON r.resident\_id = b.resident\_id

JOIN Utility\_Types ut ON b.utility\_id = ut.utility\_id

JOIN Utility\_Usage u ON b.bill\_id = u.bill\_id

GROUP BY r.resident\_id, r.name, month, ut.utility\_name;

A close up of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**(ii) vw\_high\_usage\_alerts**

CREATE VIEW vw\_high\_usage\_alerts AS

SELECT

r.name,

b.billing\_date,

ut.utility\_name,

u.usage\_amount

FROM Residents r

JOIN Bills b ON r.resident\_id = b.resident\_id

JOIN Utility\_Types ut ON b.utility\_id = ut.utility\_id

JOIN Utility\_Usage u ON b.bill\_id = u.bill\_id

WHERE u.usage\_amount > 500;

A screenshot of a computer

AI-generated content may be incorrect.

**(iii) vw\_total\_bills:**

CREATE VIEW vw\_total\_bills AS

SELECT

r.name,

COUNT(b.bill\_id) AS number\_of\_bills,

SUM(b.amount) AS total\_paid

FROM Residents r

JOIN Bills b ON r.resident\_id = b.resident\_id

GROUP BY r.name;

A screenshot of a data output

AI-generated content may be incorrect.

**Step 5: Converting SQL Schema to NoSQL (MongoDB)**

**(I) Convert Relational Structure to Embedded JSON**

Instead of normalized tables (Residents, Bills, Utility\_Types, Utility\_Usage), we embed the data. Your example JSON structure is perfect.

**Import Embedded JSON into MongoDB Atlas**

Inserting residents to create small data samples:

**Residents 1 and 2:**

A screenshot of a computer

AI-generated content may be incorrect.

**Residents 3 and 4:**

A screenshot of a computer

AI-generated content may be incorrect.

**(II) Running Queries**

Use the **Filter** box under your collection to run some queries.

**1. Find all residents using Electricity:**

A screenshot of a computer

AI-generated content may be incorrect.

**2. Find all residents with water bills:**

A screenshot of a computer

AI-generated content may be incorrect.

**3. Find a resident by name:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**4. Find residents whose usage is more than 100 units:**

**A screenshot of a computer

AI-generated content may be incorrect.**